The NIST
ATM PNNI Routing Protocol Simulator
(APRoPS)

Operation and Programming Guide

Version 2.0

Yunming Song
David Cypher
David Su

U.S. Department of Commerce
Technology Administration
National Institute of Standards and Technology
Information Technology Laboratory
Advanced Networks Technologies Division
High Speed Network Technologies Group
Gaithersburg, MD 20899-8920

September 1999

TABLE OF CONTENTS

The NIST ATM PNNI Routing Protocol Simulator

Operation and Programming Guide

Version 2.0

Abstract

An Asynchronous Transfer Mode (ATM) Private Network-Network Interface (PNNI) routing protocol simulator has been developed to provide a means for researchers and network planners to analyze the behavior of ATM network routing protocols. The simulator is a tool that gives the user an interactive modeling environment with a user-friendly user interface. With this tool the user can create different network topologies, control component parameters, measure network performance, and log data from simulation runs. This document provides instructions for creating network configurations, specifying component parameters, and controlling the display of the simulation results. The design of the simulator is also provided to guide the user, who wishes to modify the simulator source code, to accommodate network components not previously defined, or to change the behavior of components already defined.

Version 2.0 contains modifications to the user interface and additional features. Detailed modifications and additions are listed in Annex B: Modifications and additions to Version 2.0 from Version 1.1. Additionally, Version 2.0 now contains a separate graphical user interface that is described in Section 4.

1 Introduction

The ATM PNNI Routing Protocol Simulator was developed at the National Institute of Standards and Technology (NIST) to provide a flexible test bed for studying and evaluating the performance of ATM network routing based on the ATM Forum Specification, "Private Network-Network Interface Specification V1.0". The simulator is a tool that gives the user an interactive modeling environment with a user-friendly user interface (an adjunct graphical user interface is now included (see section 4)). NIST has developed this tool using the "C" programming language to execute on a SUN SPARCstation using SunOS Release 5.5.1. The adjunct graphical user interface uses the Tcl/Tk programming language. This simulation tool is based upon discrete event simulation techniques. The adjunct graphical user interface facilites the user in adding, removing or modifying nodes or links in the given network topology.

The ATM PNNI Routing Protocol Simulator allows the user to create different network topologies, set the parameters of component operation, and load the different simulated configurations. While the simulation is running, various instantaneous performance measures can be displayed in text form on the screen or the user can record these for subsequent analysis. With this simulator, the user can design and test almost any network configuration and evaluate its performance within the constraints of the physical machine running the simulation. Therefore the user can evaluate the scalability, robustness and maintainability of PNNI routing status.

2 Overview of the Simulator

There are two basic goals for the simulator: evaluating the stability of the PNNI routing protocol and studying the scalability of the PNNI routing protocol. Based upon these goals, the simulator can be used as a tool for PNNI routing network planning and for PNNI routing protocol performance analysis. As a planning or protocol performance analysis tool, a network planner, researcher or protocol designer can run the simulator with various network configurations to study the network performance. It could be used to answer the following questions:

- How much time is needed for the PNNI routing protocol to reach stability?
- How much data must be exchanged for the PNNI routing protocol to reach stability?
- How much data must be exchanged to maintain the PNNI routing status, once initial stability is reached?
- How quickly does the PNNI routing protocol recover from link or node failures?

Note that stability is defined as the state where each of the PNNI routing protocols has reached an operating mode, which is considered a final state.

The simulator is designed in such a way that modules simulating switches or links within an ATM PNNI routing network can be easily modified, added or removed.

2.1 Components

The network to be simulated consists of two *components: ATM Switches* and *Physical Links.* All components are characterized by one or more *parameters*. All parameters may be specified by the user at the time of component creation and may be modified later, through the use of the adjunct graphical user interface.

The ATM switch component simulates a PNNI capable node (i.e., it implements the protocols and Finite State Machines (FSM)). The Physical link simulates the physical medium (e.g.,

optical fiber) on which packets are transmitted. A single physical link component connects two ATM switch components.

2.1.1 ATM Switch

Another term for an ATM switch is "Node". The user must specify the following PNNI information (i.e., parameters) per node: ATM End System Address and Node level. Optionally, the user may specify non-default values for Peer Group Leadership Priority or the PTSE refresh interval.

The user must also specify the time when a node is to become active (i.e., active/up time). The user may also enter other node configuration values (i.e., a node's processing time for each of the different packet types: Hello, Database Summary, PTSE Request, PTSP, and PTSE Acknowledgment).

It is assumed that the node's buffer is unlimited, meaning that all events and packets can be stored by this node.

2.1.2 Physical Link

There are two parameters, physical link delay and active/up time, that define a physical link. The user must specify a physical link delay. The active/up time depends on two situations. The first is when the physical link becomes active when both nodes that it connects become active. In this case there is no need to specify an active/up time. The second is when the physical link is to become active at a later time. In this case the user must enter a value for the active/up time. It is assumed that the usable bandwidth of the link is unlimited.

3 Operating the Simulator

This part of the document provides information for the user of the simulator to create network topologies using simulated ATM switches (i.e., nodes), and physical links. The manual includes instructions for display manipulation, component linking, parameter setting, data output and analysis.

The user may control the parameters associated with these components, and specify many details concerning the logging and display of performance data. In this version, the user interface contains two kinds of formats. One is in text form, which is integrated with the simulator program. The other is in graphical form, which is an adjunct program for the simulator program (described in Section 4). This section only describes the text user interface.

The user should first design the network topology with the necessary parameters before starting the program. This will aid in the entering of the data into the simulator.

3.1 Starting the Program

After the "C" program of the simulator is compiled by running the "makefile"(i.e. at the command line type make or make –f makefile), one can execute the ATM PNNI Routing Protocol Simulator by typing the following at the command line:

aprops.exe *seed_value out_configuration_file* "or"
aprops.exe *seed_value output_configuration_file input_configuration_file*

This invokes the text user interface followed by the execution at the interface to create a PNNI network configuration. The *seed_value* is an integer that is used as an input parameter to the random function, srand ( ). The *output_configuration_file* is a file that will be created that will contain the user's network configuration as entered during the running of the text user interface. This helps to automate the creation of a network configuration file. The *input_configuration_file* is a file that contains an existing network configuration. This permits the user to reuse a previous network configuration without the need to reenter the data.

Note that, the *output_configuration_file* and *input_configuration_file* can not use the same name. The user can directly use the simulator program (i.e. the text user interface) to create an *output_configuration_file*. As an option, the user may first use the adjunct graphical user interface to create a *network_configuration_file*, and then use this file as an *input_configuration_file* at the command line.

3.2 User Interface

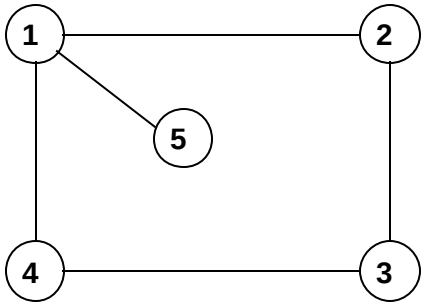
For the text User Interface (UI), it allows the user of the simulator to create the network topology and to set the parameters for simulation runs. The user is first presented with the simulator's default values, which the user is asked to accept or modify. After the defaults are set, then the user can define the network topology. The user interface then prompts for the total number of lowest level nodes, followed by questions for each node. Next the node connectivity is configured. The connectivity of nodes is processed according to the sequence in which the nodes were entered. After the network topology is entered, the user is asked to confirm the input. If the user does not accept the current configuration, the program will terminate and the user must rerun the program. If the input is acceptable, then the UI prompts for possible failures to be simulated. Next the amount of time that the program is to simulate is entered. The method of the output display is entered. Note that, due to the constraints of the text user interface, the user cannot modify the network topology during topology configuration. When using the adjunct graphical user interface, the user can easily modify the network topology (e.g. add/remove a node or link).

The UI provides many options to cover many kinds of situations. It permits the user to define the running simulation period so as to observe the running results. It also permits the user to define a link failure or node failure time and link restore or node restore time.

When defining a component for the first time, a question will appear automatically, asking for the required information. Each entry is terminated by a *Enter*. A *Enter* with no entry will accept the default value. No other action is possible until all requested information has been entered. If the entered value is not correct, a warning message will appear, and then a new value must be entered. As the following example shows:

3.2.1 An Example

The user creates the following network topology:



Assume that:

1) all network nodes and links are active at once, that is, all nodes are turned on at the same time;
2) all nodes are located in the same peer group;
3) all nodes are configured with different peer group leadership priorities;
4) all nodes have the same node processing overheads for the same packet types;
5) all links have the same link delay;
6) After simulation for a while, a certain link fails and later on it is restored.

The following text is an example of the simulator screen seen by the user when a network configuration is created.

--> **aprops.exe** *seed_value output_configuration_file*

Welcome to APROPS

APROPS will ask some questions for entering a PNNI network topology:
1) PNNI simulation option (e.g. PTSP bundling);
2) Default node processing overheads and refresh interval;
3) Node number and node configuration (e.g. node addressing, node level);
4) Connectivity configuration (e.g. links);

5) Whether to simulate a link failure or node failure and then to restore it (e.g. failure time and restore time);
6) Simulation period and output choice;
7) Details for the link failure or node failure, if any;
8) Details for restoring the link failure or node failure, if any.

The option for PTSP during database synchronization:
One PTSP which bundles all routing database information in a node (0)
One PTSP which bundles each routing database information in a node (1)
PTSP: PNNI Topology State Packet
Select (0/1: default=0) ? *Enter*
The option for PTSE refreshing:
One PTSP bundles all PTSEs in a node (0)
One PTSP bundles each PTSE in a node (1)
PTSE: PNNI Topology State Element
Select (0/1: default=0) ? 1*Enter*

Current default node processing overheads and Refresh Interval are the the following:
1)The node processing overhead for Hello packet is 0.1 second.
2)The node processing overhead for Database Summary packet is 0.3 second.
3)The node processing overhead for PTSE REQUEST packet is 0.5 second.
4)The node processing overhead for PTSP packet is 0.5 second.
5)The PTSE refresh interval is 1800.0 seconds.

Would you like to change the current default values (y/n: default=n) ? y*Enter*
The node processing overhead (default=0.1 second) for Hello packet ? *Enter*
The node processing overhead (default=0.3 second) for Database Summary packet ? *Enter*
The node processing overhead (default=0.5 second) for PTSE REQUEST packet ? *Enter*
The node processing overhead (default=0.5 second) for PTSP packet ? *Enter*
The PTSE refresh interval (default=1800.0 seconds) ? 1200*Enter*

Note that in the following inputs:
1) All ATM addresses must be unique !
Example: 0x47.00.05.80.00.5a.ff.00.00.00.02.03.7c.00.02.e1.0f.81.01.00
2) To simplify the program, the Node Level uses only multiples of 8 bits, for example, if this level is 56, the next higher level is 48.
Note that, right now the program supports only one higher level.
3) Each node can connect to at most 10 other nodes at the lowest level.
4) There can be at most 3 physical links between two nodes of same peer group at the lowest level, or at most one physical link otherwise.
5) The global clock starting point is 0.0 second.

With the above information,
what is the total number of nodes for this PNNI network topology ? 5*Enter*

ATM address of this node ? 47.0005.80005aff00000002037c.0002e10f8101.00*Enter*
Node Level ? 56*Enter*
Peer Group Leadership Priority (default=0) ? 1*Enter*
Will this node use different node processing overheads or refresh interval (y/n: default=n) ? *Enter*
When will this node become active (default=0.0 second) ? *Enter*
You have entered 1 nodes in the topology !

ATM address of this node ? 47.0005.80005aff00000002037c.0002e10f8102.00*Enter*
Node Level ? 56*Enter*

Peer Group Leadership Priority (default=0) ? 2*Enter*
Will this node use different node processing overheads or refresh interval (y/n: default=n) ? *Enter*
When will this node become active (default=0.0 second) ? *Enter*
You have entered 2 nodes in the topology !

ATM address of this node ? 47.0005.80005aff00000002037c.0002e10f8103.00*Enter*
Node Level ? 56*Enter*
Peer Group Leadership Priority (default=0) ? 3*Enter*
Will this node use different node processing overheads or refresh interval (y/n: default=n) ? *Enter*
When will this node become active (default=0.0 second) ? *Enter*
You have entered 3 nodes in the topology !

ATM address of this node ? 47.0005.80005aff00000002037c.0002e10f8104.00*Enter*
Node Level ? 56*Enter*
Peer Group Leadership Priority (default=0) ? 4*Enter*
Will this node use different node processing overheads or refresh interval (y/n: default=n) ? *Enter*
When will this node become active (default=0.0 second) ? *Enter*
You have entered 4 nodes in the topology !

ATM address of this node ? 47.0005.80005aff00000002037c.0002e10f8105.00*Enter*
Node Level ? 56*Enter*
Peer Group Leadership Priority (default=0) ? 5*Enter*
Will this node use different node processing overheads or refresh interval (y/n: default=n) ? *Enter*
When will this node become active (default=0.0 second) ? *Enter*
You have entered 5 nodes in the topology !

How many nodes are connected to specified node 47.0005.80005aff00000002037c.0002e10f8101.00 ? 3*Enter*
First connected node's ATM address ? 47.0005.80005aff00000002037c.0002e10f8102.00*Enter*
How many physical links (default=1) exist between two nodes? *Enter*
The delay (default=0.0001 second) for the physical link 1 ? *Enter*
Is the physical link connected in the initial configuration?
(That is: does the physical link 1 become active when both nodes
at the end of the physical link 1 are active/up ?) (0)
Or is the physical link 1 not currently connected, but will be some time in the future after both nodes are active, i.e., any value greater
than both node active times 0.000000 and 0.000000 (1) ?
Select (0/1: default=0) ? *Enter*
Second connected node's ATM address ? 47.0005.80005aff00000002037c.0002e10f8104.00*Enter*
How many physical links (default=1) exist between two nodes? *Enter*
The delay (default=0.0001 second) for the physical link 1 ? *Enter*
Is the physical link connected in the initial configuration
(That is: does the physical link 1 become active when both nodes
at the end of the physical link 1 are active/up ?) (0)
Or is the physical link 1 not currently connected, but will be some
time in the future after both nodes are active, i.e., any value greater
than both node active times 0.000000 and 0.000000 (1) ?
Select (0/1: default=0) ? *Enter*
Third connected node's ATM address ? 47.0005.80005aff00000002037c.0002e10f8105.00*Enter*
How many physical links (default=1) exist between two nodes? *Enter*
The delay (default=0.0001 second) for the physical link 1 ? *Enter*
Is the physical link connected in the initial configuration
(That is: does the physical link 1 become active when both nodes
at the end of the physical link 1 are active/up ?) (0)

Or is the physical link 1 not currently connected, but will be some time in the future after both nodes are active, i.e., any value greater than both node active times 0.000000 and 0.000000 (1) ?
Select (0/1: default=0) ? *Enter*

How many more nodes are connected to specified node 47.0005.80005aff00000002037c.0002e10f8102.00 ? 1*Enter*
Connected node's ATM address ? 47.0005.80005aff00000002037c.0002e10f8103.00*Enter*
How many physical links (default=1) exist between two nodes? *Enter*
The delay (default=0.0001 second) for the physical link 1 ? *Enter*
Is the physical link connected in the initial configuration
(That is: does the physical link 1 become active when both nodes at the end of the physical link 1 are active/up ?) (0)
Or is the physical link 1 not currently connected, but will be some time in the future after both nodes are active, i.e., any value greater than both node active times 0.000000 and 0.000000 (1) ?
Select (0/1: default=0) ? *Enter*

How many more nodes are connected to specified node 47.0005.80005aff00000002037c.0002e10f8103.00 ? 1*Enter*
Connected node's ATM address ? 47.0005.80005aff00000002037c.0002e10f8104.00*Enter*
How many physical links (default=1) exist between two nodes? *Enter*
The delay (default=0.0001 second) for the physical link 1 ?
Is the physical link connected in the initial configuration
(That is: does the physical link 1 become active when both nodes at the end of the physical link 1 are active/up ?) (0)
Or is the physical link 1 not currently connected, but will be some time in the future after both nodes are active, i.e., any value greater than both node active times 0.000000 and 0.000000 (1) ?
Select (0/1: default=0) ? *Enter*

How many more nodes are connected to specified node 47.0005.80005aff00000002037c.0002e10f8104.00 ? 0*Enter*

How many more nodes are connected to specified node 47.0005.80005aff00000002037c.0002e10f8105.00 ? 0*Enter*

The Topology is the following:
(This is original topology)
Node 1 (seq): 38.a0.47.0005.80005aff00000002037c.0002e10f8101.00
Node 2 (seq): 38.a0.47.0005.80005aff00000002037c.0002e10f8102.00
Node 3 (seq): 38.a0.47.0005.80005aff00000002037c.0002e10f8103.00
Node 4 (seq): 38.a0.47.0005.80005aff00000002037c.0002e10f8104.00
Node 5 (seq): 38.a0.47.0005.80005aff00000002037c.0002e10f8105.00

Nodes	1	2	3	4	5
1		*(1)		*(1)	*(1)
2			*(1)		
3				*(1)	

4
5
The '*' means that both nodes are connected;
the '(1/2/3)' is the number of physical links between two nodes.

A configuration file has been created. If you want to change any value,
you must rerun the program. If all values are correct, then continue !
Do you want to rerun (r) or are all values correct (c) (default=c) ? *Enter*
Do you want to simulate any link failure or node failure (y/n: default=n) ? y*Enter*
When does the failure take place (seconds) ? 200*Enter*
Do you want to simulate a physical link failure (l) or node failure (n) (default=n) ? l*Enter*
The two nodes for failed physical link are the following:
First node's ATM address ? 47.0005.80005aff00000002037c.0002e10f8102.00*Enter*
Second node's ATM address ? 47.0005.80005aff00000002037c.0002e10f8103.00*Enter*
There is 1 physical link(s) between these two nodes !

Do you want to restore the link failure (y/n: default=n) ? y*Enter*
When is the failure restored (seconds) ? 450*Enter*
The restored link is located between the two nodes:
47.0005.80005aff00000002037c.0002e10f8102.00
47.0005.80005aff00000002037c.0002e10f8103.00

How much time do you want the program to simulate (seconds) ? 600*Enter*
Do you want to output data count per link each time (y/n: default=n) ? *Enter*
(This will give user the results about transmitted data for different packets or total amount at both directions of each link at any event execution time or at the end.)

The simulation results of program are the following:

……

!!!The PNNI has reached stability of database synchronization!!!
for the following nodes:
(Nodes are the same as before)
The data (bytes) transmitted until 3.601101 seconds are the following:
From node to node: Hello, DBS, REQ, PTSP, ACK, Total-Hello, Total
1 to 2 (link 1): 318, 212, 53, 2226, 636, 3127, 3445
2 to 1 (link 1): 318, 212, 53, 1484, 954, 2703, 3021
1 to 4 (link 1): 318, 212, 53, 2226, 636, 3127, 3445
4 to 1 (link 1): 318, 212, 53, 1484, 954, 2703, 3021
1 to 5 (link 1): 318, 212, 53, 3233, 212, 3710, 4028
5 to 1 (link 1): 318, 212, 53, 477, 1272, 2014, 2332
2 to 3 (link 1): 318, 212, 53, 2226, 636, 3127, 3445
3 to 2 (link 1): 318, 212, 53, 1484, 954, 2703, 3021
3 to 4 (link 1): 318, 212, 53, 1484, 954, 2703, 3021
4 to 3 (link 1): 318, 212, 53, 2226, 636, 3127, 3445
The total (total-hello) data (bytes) and average (bytes/second/link) are:
32224 (29044) and 1789.674880 (1613.062227) !

!!!The PNNI has reached stability of PGLE !!!
for the following nodes:
(Nodes are the same as before)
The data (bytes) transmitted until 53.623013 seconds are the following:

From node to node: Hello, DBS, REQ, PTSP, ACK, Total-Hello, Total
(leave out)
The total (total-hello) data (bytes) and average (bytes/second/link) are:
46693 (38743) and 174.152841 (144.501393) !

……
(simulation until 200 seconds)

Current topology is the following:
(same as original one)

The two nodes for failed physical link 1 are the following:
47.0005.80005aff00000002037c.0002e10f8102.00
47.0005.80005aff00000002037c.0002e10f8103.00

……..…….
(simulation until 450 seconds)

Original topology is the following:
(leave out)

New/Current topology with link failure or node failure is the following:

Nodes	1	2	3	4	5
1		*(1)		*(1)	*(1)
2					
3				*(1)	
4					
5					

The restored link 1 is located between two nodes:
47.0005.80005aff00000002037c.0002e10f8102.00
47.0005.80005aff00000002037c.0002e10f8103.00

…….

!!!The PNNI has reached stability of database synchronization!!!
for the following nodes:
(Nodes are the same as before)
The data (bytes) transmitted until 452.900901 seconds are the following:
From node to node: Hello, DBS, REQ, PTSP, ACK, Total-Hello, Total
(leave out)
The total (total-hello) data (bytes) and average (bytes/second/link) are:
93174 (47382) and 41.145425 (20.923783) !

……..…….
(simulation until 600 seconds)

3.3 Operating the Simulator

3.3.1 Loading a Network Configuration

There are three ways to specify a network configuration for the program to simulate.

1. Create a network configuration while in the simulator program using the text user interface (as previously described). Using this approach, the user decides on the appropriate components, their characteristics and interconnections.

2. Reuse an existing network configuration by using one of the following two methods.
   i) access
   aprops.exe *seed_value output_configuration_file input_configuration_file*

   ii) redirect
   aprops.exe *seed_value output_configuration_file* < *input_configuration_file*

The input configuration file can be created in one of three ways:
1) by directly coping the output_configuration_file from bullet item #1 above;
2) by manual entry of an ASCII file by a user who is very familiar with the usage of the simulator program, or
3) by using the output_configuration_file from the adjunct graphical user interface (see section 4).

A sample input_configuration_file follows. Each line of this configuration file contains a response for each question asked by the User Interface.

Note: When the user uses the redirection method, if the name of the *output_configuration_file* already exists, the redirection method will fail. The *output_configuration_file* must be renamed and then tried again.

Note: The input_configuration_file cannot contain any comments.

```
0
1
y
0.1
0.3
0.5
0.5
1200
5
47.0005.80005aff00000002037c.0002e10f8101.00
56
```

1
n
0.0
47.0005.80005aff00000002037c.0002e10f8102.00
56
2
n
0.0
47.0005.80005aff00000002037c.0002e10f8103.00
56
3
n
0.0
47.0005.80005aff00000002037c.0002e10f8104.00
56
4
n
0.0
47.0005.80005aff00000002037c.0002e10f8105.00
56
5
n
0.0
3
47.0005.80005aff00000002037c.0002e10f8102.00
1
0.0001
0
47.0005.80005aff00000002037c.0002e10f8104.00
1
0.0001
0
47.0005.80005aff00000002037c.0002e10f8105.00
1
0.0001
0
1
47.0005.80005aff00000002037c.0002e10f8103.00
1
0.0001
0
1
47.0005.80005aff00000002037c.0002e10f8104.00
1
0.0001
0
0
0
c
y
200
l
47.0005.80005aff00000002037c.0002e10f8102.00
47.0005.80005aff00000002037c.0002e10f8103.00

y
450
600
n

3.3.2 Creating a Network Configuration

Creating a network configuration starts with a written network configuration plan, which is based on the total number of nodes at the lowest level.

3.3.2.1 **Creating Defaults.**

This simulation has some built in defaults, which by default apply to the entire network configuration. The initial defaults are PTSP bundling, node processing overheads for different types of packets, PTSE refresh interval, and physical link delay. These defaults may be changed to apply to some or all of the components in a network configuration.

3.3.2.2 **Creating Components.**

There are two types of components: node and link. Nodes are created when the user is asked for the total number of nodes at the lowest level. For each node, the user is in turn prompted to enter values for each parameter, including ATM address, node level, peer group leadership priority, and node active time, which will specify the component's characteristics in the network. After all the required parameter values are entered, the component will be created. The relative location of the component in the network will be determined by the connectivity between this component and other components.

3.3.2.3 **Linking Components.**

After the node components have been created, they are connected by creating physical links. This forms a network. A physical link is a network component with associated parameters. A physical link is specified by the two nodes that are connected by that link. Meaning that for each node, the user must enter the other nodes to which this node is connected and then enter the number of links connecting these two nodes.

However, the user needs to know the network topology. After making sure that there is a link between two nodes, and all the required parameter values of the link are entered, the link will be created.

3.4 Operational Features

This simulator provides several features, which may be used to enhance the display of information and to modify the network that was created.

3.4.1 Displaying Information about the Network

There are two methods for outputting the display information. One lists the executed events followed by an output summary. The other lists an output summary for each execution of a group of events.

3.4.2 Making Modifications

For the text user interface, the only way to make modifications for the beginner is to rerun the program. If the user has specified a configuration file (i.e., ASCII file), it is easier for the user to modify a few parameters, which depended on the experience. If the user is familiar with the text user interface and the configuration file, it is not so difficult for the user to modify some of the parameters of the PNNI network topology. For example, if the user wants to modify the peer group leadership priority for a certain node, the user can directly search for the node and modify it. The same is true for the ATM address, and node level. Note that, the modification of the ATM address must be modified in all other relevant places. If the user wants to simulate a link failure and then restore it, the user only needs to specify several lines, for instance:

```
c
y
200
l
47.0005.80005aff00000002037c.0002e10f8102.00
47.0005.80005aff00000002037c.0002e10f8103.00
y
450
600
```

The failure and restore configuration contains 7 lines between the flag, "c", and simulation period, "600". For any other link failure, the user only needs to modify the ATM addresses plus the link sequence if there exists more than one physical link. If the user specifies a node failure, it will have the following format:

```
c
y
200
n
47.0005.80005aff00000002037c.0002e10f8102.00
p
y
450
600
```

Nevertheless, the adjunct graphical user interface should be used when making modifications, since it makes it much easier for the user to add, remove or modify any nodes and links.

3.4.3 Processing of Partitions

Normally there is no partitioning within a peer group. However, the simulator program provides the mechanism and capability to process two or more partitions for one peer group due to a link failure or node failure. In each partition, the database synchronization and PGL election will be preformed. If the partitioned peer group has an outside link connecting with other peer groups, it may perform higher level protocols, including the SVCC-based RCC Hello protocol, SVCC-based RCC DBS protocol and LGN Horizontal link Hello protocol, etc., which depends on their respective connectivity with other peer groups.

3.4.4 Determination of Stability

Using the simulator program, the user can do many evaluations, which include the time it takes for the PNNI routing protocol to reach stability and the amount of data required for PNNI operations alone, both in reaching stability and for maintaining stability. As seen in the output of the above simulation example, the determination of stability is an important feature, which gives the user a general vision of the program running. With this capability, the user can easily find when the protocol will reach stability and how much data is needed to reach stability. Also it is easy to determine the maintainability based upon reaching stability.

For the lowest level, the stability includes the completion of database synchronization and PGL election. For the higher level, the stability only includes the completion of database synchronization. If there is a link failure or node failure, the PNNI routing protocol may reach again the stability of database synchronization or PGL election, that depends on the connectivity among all nodes within one peer group and whether the failed node is just a PGL for one peer group, etc. The peer group may be partitioned due to the link failure or node failure, which may invoke more than one SVCC-based RCC existing between two adjacent peer groups and the determination of stability will be more complicated.

4 Adjunct Graphical User Interface

The purpose of this adjunct graphical user interface is to provide an easier method for entering the network configuration and to provide an automated method for modifying existing network configurations.

4.1 Introduction

The Adjunct Graphical User Interface (GUI) for the simulator program (APRoPS) has been developed by using the Tcl/Tk programming language running on a SUN SPARCstation platform using SunOS Release 5.5.1 Generic. Basically, the GUI is composed of the following windows in turn:

- A file name window is used to ask the names for the output configuration file and the input configuration file, if any.

- An option window is used to setup the options for PTSP bundling during database synchronization and for PTSE refreshing.

- A default value window is used to setup the default node processing overheads and the refresh interval for all nodes.

- A node configuration window is used to setup ATM address, node level, peer group leadership priority, and node active time.

- A connectivity configuration window is used to setup the connectivity, i.e. physical links for all nodes.

- A modification window is used to add, remove and modify any node or any link, if any.

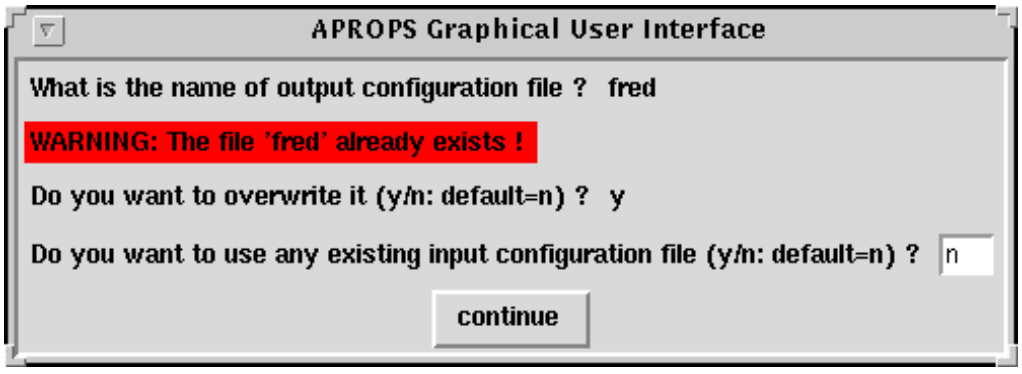
- A failure window is used to detail the link failure or node failure information, including failure time, failed link and link sequence, etc.

- An output window is used to setup the simulation period and output option.

4.2 Operating the GUI

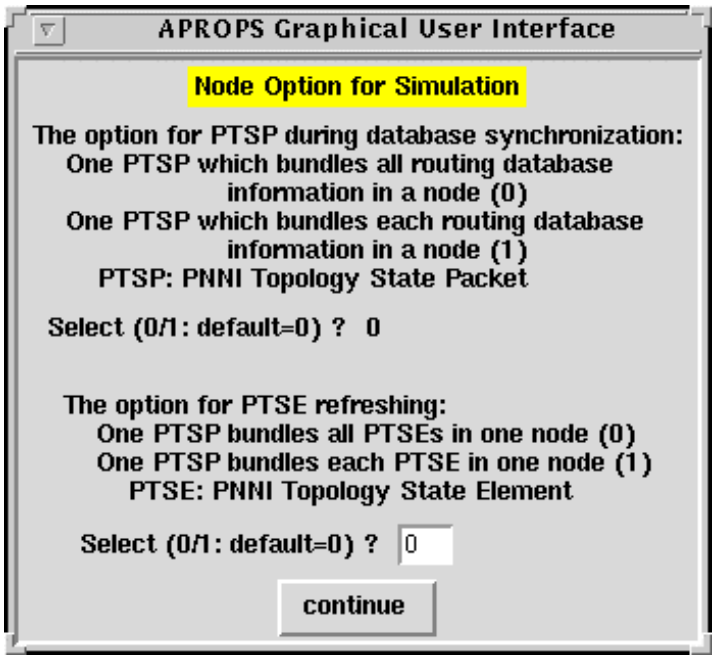
Please make sure that you have a version of TCL/TK (at least version 4.2) installed on your SunOS and it is in your path, then run the TCL/TK program, *interface*, at the command prompt.

4.2.1 The File Name Window



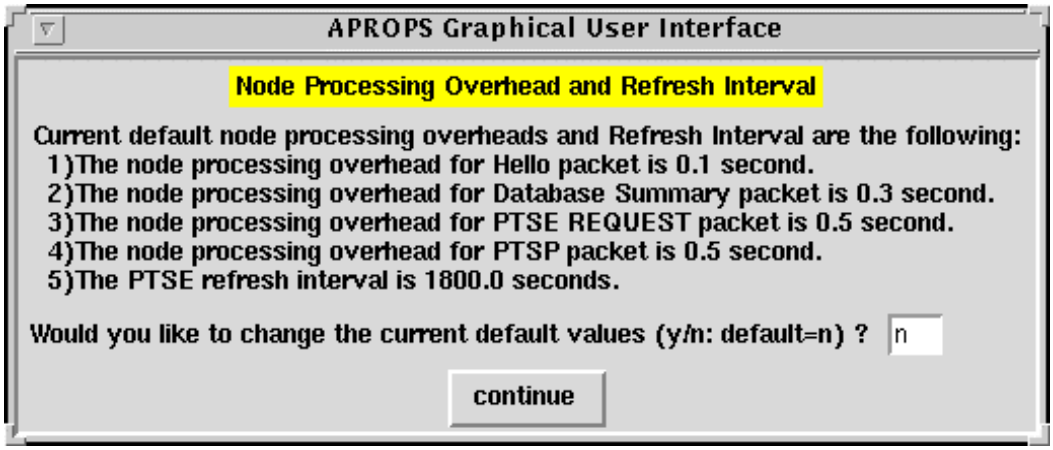
The first window prompts the user for the name of the output configuration file, which is to be created. If the user types in a name of which a file already exists by that same name, the GUI will show a warning message (as shown above) and asks whether or not it is to be overwritten. The user is then asked whether or not to use an existing input configuration file. If yes, then the user must enter the name of the existing input configuration file. Finally push the "continue" button to go to the next question or the next window.

4.2.2 The Option Window



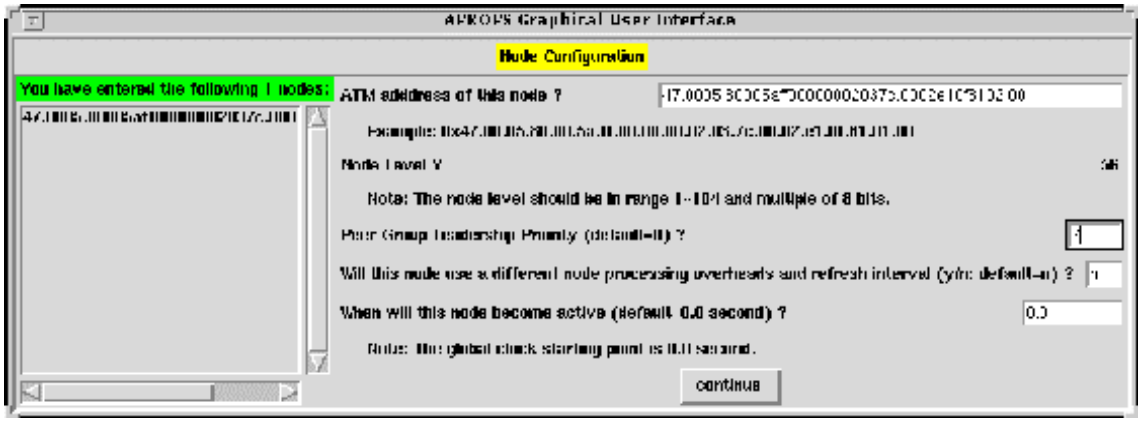
The user selects the options according to the practicular simulation requirements. The default selection is "0". The figure shows that the default was accepted for both questions.

4.2.3 The Default Value Window



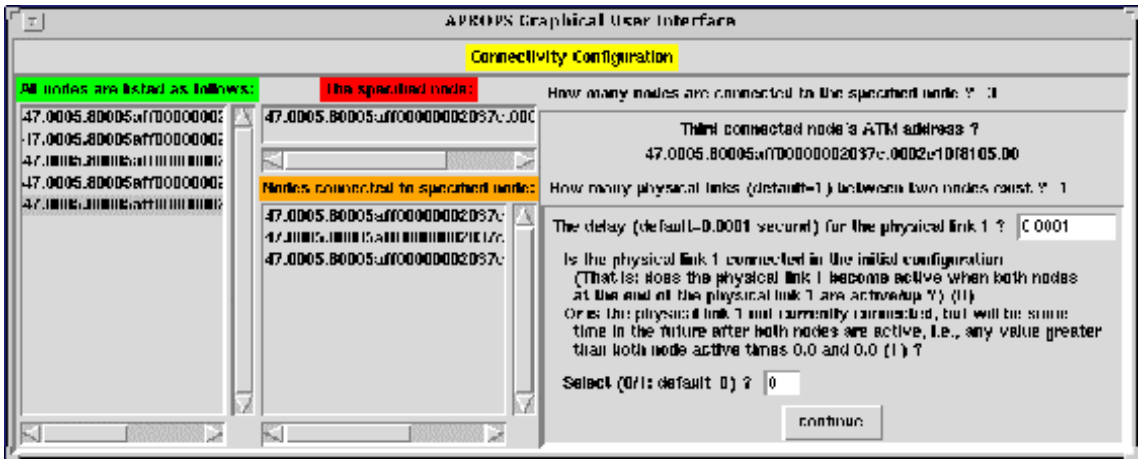
All of the above default values were obtained from the practical measurements for certain ATM switch systems. The user can modify these values accordingly. If the user answers yes, then another window is opened to change values. Figure shows the acceptance of the default values for these parameters.

4.2.4 The Node Configuration Window



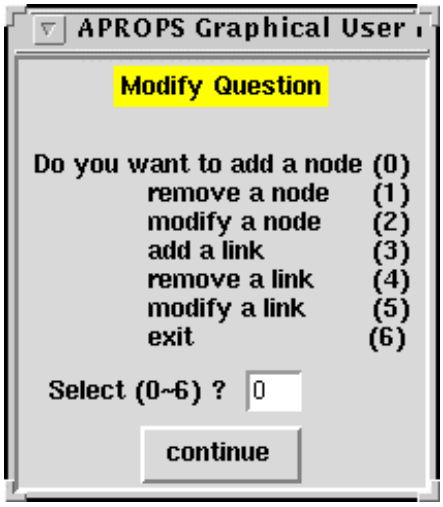
In this window, the left box lists all of the nodes the user has entered, and the right side items are the current node's values, that the user will enter. For the user's convenience, once a node's ATM address has been entered and appears in the box on the left, it can be copied. The user can click on a previous ATM address from the left box. The clicked ATM address will show up at the right side, and then the user only needs to modify the related bytes.

4.2.5 The Connectivity Configuration Window



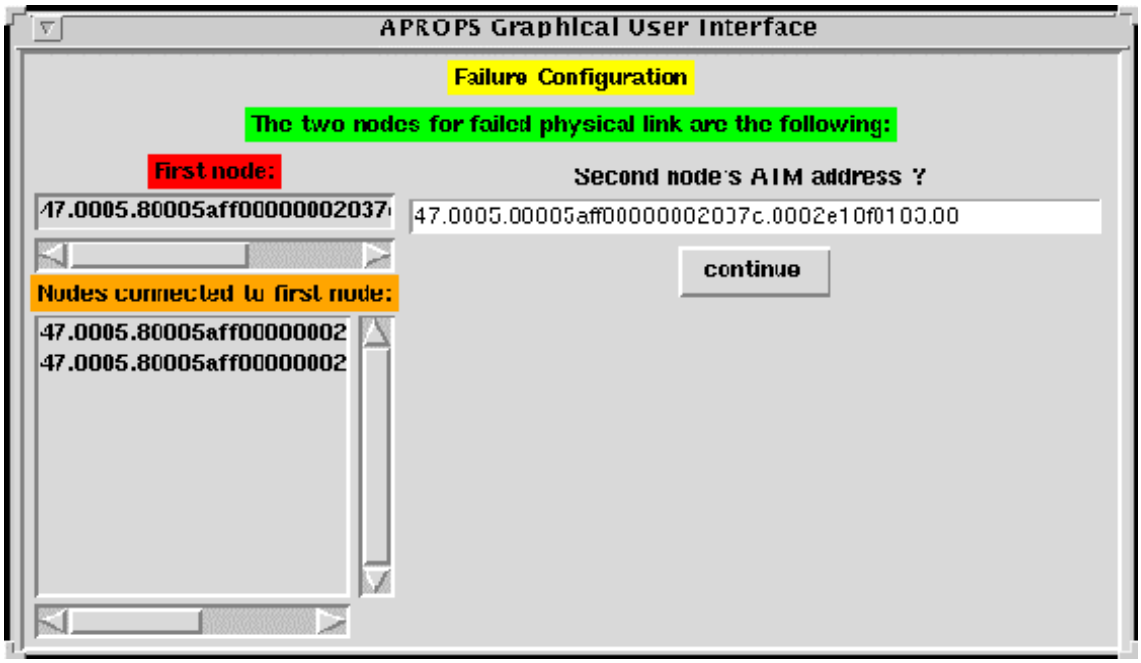
In this window, the leftmost box lists all of the entered nodes in the PNNI network topology. The specified node means the node currently asked for the connectivity. Shown below the specified node's ATM address are all of the other nodes that are already connected to the specified node. The right side asks about the physical link delay and link active time for this connection.

4.2.6 The Modification Window



The user can choose from this list to make any modification to the existing PNNI network topology. There is no limit to the number of modifications that can be preformed at this time. This window is only displayed after the user has completed entering a configuration and the user has selected to modify the current configuration, instead of accepting the configuration as is.

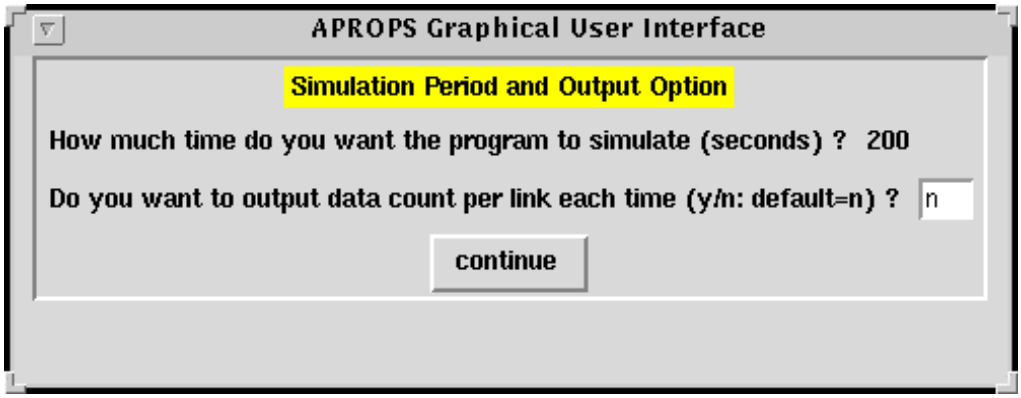
4.2.7 The Failure Window



First of all, the link failure or node failure is asked only when the network topology configuration is finished. As seen in this window, after the first node is selected, the choice of the second

node is limited to one of those, which have connections with the first node. For the user's convenience, the user does not need to manually type in the second node's ATM address. The user can click on the selected ATM address from the "Nodes connected to first node" box.

4.2.8 The Output Window



In this window, the user needs to specify the simulation period and output option. However, the simulation period must be greater than the failure time and restore time, if any. If the user enters a wrong value, the GUI will give the user a warning message, and provide an entry so that it may be corrected.

5 Simulator Concepts

5.1 Simulator Clock

The simulator is *discrete-event* driven, meaning that the events of interest occur at discrete points in time. Under a discrete-event model, the time parameter is conceptually continuous and can thus be arbitrarily increased. The discrete changes in states or events occur at any real time. The simulation software contains an event manager (EM), which provides a general facility to schedule and send or fire an event. The function of the EM is to maintain and update an event queue consisting of events that are arranged in time order of execution, including the timer expiration events (e.g. Hello timer and Inactivity timer). The EM also maintains a global clock to keep track of the simulation time. The accuracy of the global clock is predefined in the simulator program. The time accuracy is used to determine whether two or more of the time stamped values for the given events are to be considered the same time. A cycle of discrete-event simulation is depicted by the following steps:

1) When the clock is just not earlier than the earliest occurrence time, the EM selects the events with the earliest occurrence time in the event queue, including the timer

expiration events. Note that there may be more than one selected event with the earliest occurrence time within the time accuracy.

2) The EM fires the selected events. When the selected events are fired they are removed from the event queue, and the related event routines are executed. Furthermore, firing the events may create one or more future events, which are then inserted at the appropriate locations in the event queue arranged in time order of execution.
3) The global clock hops to the next earliest occurrence time, that is, the time stamp for the first event in the event queue. This approach is used to save CPU resources. The heart of the simulator is a loop executing the above cycle until the simulation is terminated.

5.2 ATM Switch

The ATM switch is the component that runs the PNNI protocols (i.e., FSMs). It sends or floods events over physical links. The firing of the event and the storing or sorting of the future event(s) is completed by the Event Manager. Except for the necessary node processing overheads based upon the network configuration, it is assumed that the switch can immediately execute the event as soon as the switch accepts the event without any additional delay (e.g., even if the switch is busy processing other tasks and the CPU resources are limited).

5.3 Link Components

A link component simulates the physical medium (e.g., optical fiber) on which PNNI packets/messages are transmitted. The user may specify the physical link delay and link active time. An important feature for the link component is to demonstrate that there exists connectivity between any two switches. If there is no link between any two switches, then there is no direct message exchange between these two switches.

5.4 Component Failure/Restore

Failures of a real network, such as link or node failures, are simulated as component failures. A component failure means that the component is simulating no activity. The component failure may be a link failure or node failure. The component failure time is the time when the component begins to fail. The component restore time is the time when the component is restored to an active status.

A failure at a node or a link (at any hierarchical level) will always cause a detectable time-out at the other node waiting for processing with the failed node/link.

- A link failure will cause the “Link Down” event at both nodes of the link.

- A physical node failure will cause the “Link Down” event at opposite sides of all the attached links.
- A logical node failure will be detected after the “Inactivity Timer Expired” event at the opposite nodes of all the attached links.

Later on, the link restore will cause the “Link Up” event at both sides of the link. The node restore will cause the “Link Up” event at both sides of all links attached to the restored node.

For the above failure, the user only needs to specify the lowest level link failure or node failure. If existing higher level connections and the failed link or node is among the corresponding SVCC-based RCC, the link failure or node failure may invoke the SVCC-based RCC unusual release, i.e. higher level “Link Down” events, and/or LGN horizontal link down. However, this is different from the lowest level link down case, except for the corresponding outside link down, there is still another choice to reestablish the SVCC-based RCC through other routes before really releasing the original one. If the failed node is just the PGL for one peer group, there is a LGN failure (higher level logical node failure). After reelecting a new PGL for the peer group, a new LGN is generated, which may create a new SVCC-based RCC with adjacent LGNs. However, two or more partitions may exist due to either a link failure or node failure, which may generate two or more LGNs for one peer group.

Note: The user can only specify one failure at one time.

5.5 Output Results

The following text is an example of the simulator screen seen by the user when the simulation is running or terminated:

Node 1 (seq): 38.a0.47.0005.80005aff00000002037c.0002e10f8101.00
Node 2 (seq): 38.a0.47.0005.80005aff00000002037c.0002e10f8102.00
Node 3 (seq): 38.a0.47.0005.80005aff00000002037c.0002e10f8103.00
Node 4 (seq): 38.a0.47.0005.80005aff00000002037c.0002e10f8104.00
Node 5 (seq): 38.a0.47.0005.80005aff00000002037c.0002e10f8105.00
The data (bytes) transmitted until 149.393393 seconds are the following:
From node to node: Hello, DBS, REQ, PTSP, ACK, Total-Hello, Total
1 to 2 (link 1): 1908, 212, 53, 2968, 848, 4081, 5989
2 to 1 (link 1): 1908, 212, 53, 1802, 1378, 3445, 5353
1 to 4 (link 1): 1908, 212, 53, 3127, 848, 4240, 6148
4 to 1 (link 1): 1749, 212, 53, 1802, 1484, 3551, 5300
1 to 5 (link 1): 1908, 212, 53, 3869, 424, 4558, 6466
5 to 1 (link 1): 1749, 212, 53, 901, 1802, 2968, 4717
2 to 3 (link 1): 1749, 212, 53, 3127, 848, 4240, 5989
3 to 2 (link 1): 1749, 212, 53, 1802, 1484, 3551, 5300
3 to 4 (link 1): 1908, 212, 53, 2385, 1484, 4134, 6042
4 to 3 (link 1): 1908, 212, 53, 3127, 1166, 4558, 6466
The total (total-hello) data (bytes) and average (bytes/second/link) are:
57770 (39326) and 77.339431 (52.647576) !

The output data is given in terms of bytes transmitted per link at a certain point in time. The link information for each node pair is split into unidirectional output. Each node pair may also have multiple links (i.e., parallel physical links). This is shown by the "(link x)". The "(link 1)" means the first parallel physical link between two nodes. The data is grouped into the type of data transmitted. Hello is the amount of data transmitted in Hello packets. The DBS, REQ, PTSP, and ACK are the amount of data transmitted in Database Summary packets, Request packets, PTSP packets, and Acknowledgement packets, respectively. The Total is the total amount of data from all the PNNI routing packets sent. The "Total-Hello" is the Total amount of data minus the amount of data generated by the Hello packets (Total column minus Hello column equals Total-Hello column).

6 Overview of the Simulator for the Programmer

This part of the document briefly describes the ATM PNNI Routing Protocol Simulator Software. It is assumed that the reader is familiar with the "C" Language programming techniques, conventions, and notations, and has the source code of the ATM PNNI Routing Protocol Simulator available.

The simulator program includes a text User Interface that provides the user with a means to define the parameters and connectivity of the network, log data, to save and load the network configuration, and to display the topology of the network. In addition to the user interface, the simulator has an event manager that can be used to schedule events. The execution of an event will cause actions or future events to happen, which are modeled separately by a number of protocol modules. The model being simulated and the actions of the events are entirely determined by the PNNI routing protocol specification, not by the framework of the simulator. The person who implements the protocol modules can decide how they will specify the PNNI routing protocol; the simulator framework only provides the means to schedule events and to communicate with the user.

The simulator software consists of:

- The source code of the simulator is written in several files (defined as **aprops.c**, **em.c**, **error.c**, **l_hello.c**, **l_dbs.c**, **l_flooding.c**, **l_pgle.c**, **l_refresh.c**, **l_failure.c**, **h_hello.c**, **h_dbs.c**, **h_flooding.c**, **h_horizontal.c** and **h_refresh.c**).
- All preprocessor macros and all data structures are globally used as a whole (defined in **aprops.h**).

The simulator software operates under the design criteria that:

- There is no limitation on network topology size, that is, the user can specify the network topology as large as he wants within the constraints of the hardware (i.e., memory).

- There is no limitation for the event queue no matter how many events are generated by the various components except for the constraints of the hardware (i.e., memory).

7 Assumptions for the Simulator

The following assumptions were made.

7.1 Assumptions to simplify the ATM PNNI simulator

- Traffic Characteristics
  - Only PNNI routing messages are studied. No other traffic is considered to be present.
  - No errors (e.g., bit errors), except for the pre-defined events (i.e., link or node failure), are considered.

- Network Hierarchy
  - The lowest level nodes are physical nodes configured with distinct addresses. The possible highest and lowest hierarchical level of a node can be determined by the node's configuration, that is, ATM address and node level.
  - The simulator only supports two levels of routing hierarchy: the lowest level and one higher level.
  - The supported number of nodes in one peer group is at most 100.
  - The supported number of lowest level peer groups is at most 20.

- Packets
  - The contents of Hello, Database Summary, PTSE Request, PTSP and PTSE Acknowledgment packets are ignored.
  - The packet sizes and PTSE sequence number are considered in order to calculate the amount of data transmitted and differentiate the PTSEs. The initial database for each node contains the Nodal Information Group (IG) and Internal reachable ATM address IG with sequence number 1. At the lowest level, each Link IG includes the linked remote node and sequence number.

- Node Connectivity
  - A single node is allowed to have connectivity to at most 10 other nodes.
    - There can be at most 3 physical links for any given node pair within one peer group, or at most one physical link otherwise.

- Peer Group Connectivity

- The peer group connectivity is determined by the network hierarchy and node connectivity. However, the supported number of node pairs as outside link between two peer groups is at most 5, and the supported number of node pairs as outside link for one peer group connecting to other peer groups is at most 10.

- Timers
  - All timers are initially disabled, i.e., they are set to 0.0.
  - Not all of the PNNI timers are used, for example, the Peer Delayed Ack Timer and the Request Rxmt Timer because there are no unexpected events taking place during the program simulation.

- Path Selection
  - The specific path selection and its related mechanisms are not considered, nor is the routing table automatically modified according to any topology change. For determining the connectivity to the PGL in one PG or determining the connectivity between two PGLs (i.e., LGNs at the higher level) in two PGs, a simple link state protocol, i.e., Shortest Path First (SPF) algorithm, is used.
  - The delay of a higher level logical link is calculated by accumulating delays along the lowest-level physical links determined by the SPF algorithm, the cost of the physical links is ignored, and their usable bandwidth is not limited.

- Failures
  - The simulator program only supports one failure (node or link) and restore per simulation run.

- SVCC Establishment Time
  - In this simulator program, the signaling protocol is not specified, therefore we assume an amount of time for the SVCC establishment after two PGLs are elected and both uplink information are received. This amount of time consists of higher level connection delay, which is the sum of related lowest level physical link delays, and higher level node processing time, which is the sum of related lowest level node processing times.

- Save Space
  - The events and states related to different protocol modules are represented by a single letter (e.g., "**A**" presents "Link Up" event in Hello FSM or "Starting" state in PGLE FSM).

7.2 General Assumptions

- The notions of the Hello protocol, the PTSE/PTSP message flooding, link aggregation, PGL election, the Topology database and the LGN Hello protocol are maintained.
  - The initial state of the Hello FSM is "Down", i.e., the link is not usable.
  - The initial state of a neighboring peer FSM is "NPDown", i.e., there are no active links to the neighboring peer.
  - The initial state of the PGL Election FSM is "Starting", i.e., the PGL Election FSM is instantiated before the first Hello FSM is started.
  - The initial state of the LGN Horizontal Link Hello FSM is "Down", i.e., no uplink PTSEs have been received including an uplink to the neighboring peer LGN with the same aggregation token value as that indicated in the LGN horizontal link hello data structure.

- Different physical links can have different link delays. Different nodes can have different packet processing overheads. A node's processing overhead can also be different for each type of packet (Hello, PTSE, DBS, etc.). Different nodes can have a different PTSE refresh interval.

- One PTSP can bundle either all routing database information stored in a node or each routing database information stored in a node. Either a single PTSE or all PTSEs in one node are flooded or refreshed at one time. The PGL can be dynamically elected, the election process only exists at the lowest level.

- A failure at a node or a link (at any hierarchical level) will always cause a detectable time-out at the other node waiting for processing with the failed node/link.
  - A link failure will cause the "Link Down" event for both nodes at the ends of the link.
  - A physical node failure will cause the "Link Down" event at the nodes at the opposite ends of all the attached links.
  - A logical node failure will be detected by the "Inactivity Timer Expired" event at the nodes at the opposite ends of all the attached links.

  The link restore will cause the "Link Up" event at both nodes attached by the link. The node restore will cause the "Link Up" event at both sides of all links attached to the restored node.

8 Components Programming

A component is a basic executing element of the simulator. There are two classes of components: ATM switches and physical links. The ATM switches allow different configurations within the network topology in order to accommodate the simulation of a variety of implementations. The simulator program is executed based upon events generated by

components, especially by the ATM switch. The event is fired based upon different protocol modules, which will be described below.

All components (ATM switches or physical links) have the same data structures that are used to store any information needed by the component, as well as standard information needed by the simulator for other components. Component information is kept in a linked list; the order of the list depends on the order of the creation of the component.

8.1 ATM switch

For the reader's convenience, the data structure used for the ATM switches is listed below:

```
struct Node {
        char PGLE_state;                    /* PGLE state */
        char address1[char_len1];           /* ATM address */
        int address[add_len];               /* digital values of ATM address */
        int level;                          /* node level */
        int priority;                       /* PGL priority */
        int nodes[node_s];                  /* all nodes in this node's partition/peer group */
        int node_number;                    /* node number in this node's partition/peer group */
        int part_seq;                       /* partition sequence within this node's peer group */
        int temp_nodes[node_s];             /* all nodes in this node's partition when existing failure */
        int temp_node_number;          /* node number in this node's partition when existing failure */
        int temp_part_seq;        /* partition sequence in this node's partition when existing failure */
        int pg_seq;                         /* this node's peer group sequence number */
        int node_seq;                       /* node sequence number */
        int LGN_ID;                         /* Logical Group Node ID */
        int attached_number;                /* connection number attached to this node */
        int attached_num;                   /* connection number attached to this node if
                                                   existing logical node failure */
        int connections[connect_s];         /* connections attached to this node */
        int node_attached[connect_s];       /* nodes attached to this node */
        int first;                          /* flag for first Hello state machine started */
        int result[node_s][data_s];         /* routing database information */
        int PGL1[node_s];                   /* PGL information for each node in this node's
                                                   partition/peer group */
        int PGL;                            /* PGL elected by this node */
        int PGL_priority;                   /* PGL's priority after PGL is elected */
        int uplink[8*outside_s1];           /* uplink PTSE information */
        int stable_nodes[node_s];  /* stable nodes for database synchronization */
        int stable_count;                   /* number of stable nodes for database synchronization */
        int stability[2];             /* flag to reach stability for database synchronization and PGLE */
        int change_flag;                    /* flag for changing preferred PGL */
        double active_time[connect_s];      /* link active time, i.e. link PTSE is generated */
        double up_time;                     /* node active/up time */
        double processing_time[4];    /* node processing overheads for different packet types      */
        double PTSERefresh_I;               /* PTSE refresh interval */
        double PGLE_timer[4];               /* PGLE timer */
        double PTSE_refresh[timer_s][3];  /* PTSE refresh  and expired timer */
```

```
        double uplink_refresh[outside_s1][3];       /* uplink PTSE refresh and expired timer */
        double PGLE_timer1[4];                      /* PGLE timer for backup */
        double PTSE_refresh1[timer_s][3];           /* PTSE refresh  and expired timer for backup */
        double uplink_refresh1[outside_s1][3];      /* uplink PTSE refresh and expired timer for
                                                                                    backup */
        struct Node *next;                          /* linked list */
} *first_node, *current_node, *node_ptr1, *node_ptr2, *node_ptr3, *node_ptr4, *node_ptr5,
*node_ptr6;
```

The network topology may have as many nodes as needed. They are stored in a linked list pointed to by **current_node**. The first node is pointed to by **first_node**, next node is pointed to by **next**, and so on. That is, each node's pointer points to next node's pointer, if a node is added, the simulator program must create and initialize a node structure and put it on its linked list. Except for the first node, all other nodes are not set to a fixed pointer because the number of nodes for each network topology is not fixed. Note that, the pointer **node_ptr1**, **node_ptr2**, **node_ptr3**, **node_ptr4**, **node_ptr5** and **node_ptr6** are used to temporarily differentiate some nodes so as not to be confused with other nodes.

In the above data structure, some parameters are explained as follows:

char_len1 represents the maximum number of characters in one line for ATM address input.

add_len represents the number of bytes for ATM address.

node_s represents the maximum number of nodes in one peer group.

nodes[node_s] used to record all nodes in this node's partition or peer group for PGLE, stability determination, etc.

part_seq used to record the partition sequence within this node's peer group. Normally, there is one partition for one peer group.

temp_nodes[node_s] used to record all nodes in this node's partition when existing link or node failure.

temp_part_seq used to record the partition sequence in this node's partition, because a link failure or node failure may lead to several partitions for this node's peer group.

connect_s represents the maximum number of connections to other nodes.

attached_num represents the connection number attached to this node if existing logical node failure, which is used to verify the failure.

data_s represents the routing database information size for each node.

PGL1[node_s] represents the PGL information for each node in this node's partition/peer group, which is used for PGLE and stability determination.

active_time[connect_s] used to record the link active time when the link PTSE is generated.

processing_time[4] used to record the node processing overheads for different packet types, including Hello, Database Summary, PTSE Request, PTSP packets.

PGLE_timer[4] used to represent the SearchPeer Timer, PGLInit Timer, OverrideUnanimity Timer and ReElection Timer for this node.

timer_s represents the maximum number of PTSE refresh timers (Nodal IG, IRA IG and Links IG, etc.).

outside_s1 represents the maximum number of node pairs as outside link for one peer group.

PTSE_refresh[timer_s][3]/uplink_refresh[outside_s1][3]

Specifically, PTSE expired timer is considered because there may exist a link or node failure in the PNNI network topology. It consists of one of two situations. The first is to represent when the PTSE expired timer for a failed node or link because the failed node or link's information may be stored in other nodes. The second is to represent the PTSE expired timer for a normal node or link, because any link or node failure may lead to lose of connectivity between any two nodes, and one node's database information may not be able to be flooded to other nodes. Note that, PTSE timer expired event only happens when there exists a link or node failure.

Note: The timer backup information is considered because the timer may be reset at some time and the original timer event is updated.

8.1.1 Higher level LGN

For the reader's convenience, the data structure used for the higher level LGN is listed below:

```
struct LGN {
        int address[add_len];                   /* ATM address */
        int level;                              /* node level */
        int nodes[node_s];                      /* all nodes in this peer group */
        int node_number;                        /* node number in this peer group */
        int PGL[part_s];            /* PGL for each partition in this peer group */
        int partition;                          /* partition number for this peer group */
        int temp_partition;          /* partition number for this peer group when existing failure */
        int pg_seq;                             /* peer group sequence number */
        int LGN_ID;                             /* Logical Group Node ID */
        int attached_number;                    /* connection number attached to this LGN */
        int attached_num;                       /* connection number attached to this LGN if existing
                                                                        logical node failure */
        int connections[h_connect_s];                   /* connections attached to this LGN */
        int lgn_attached[h_connect_s];                  /* LGNs attached to this LGN */
        int result[part_s][pg_s][h_data_s]; /* routing database information */
        int stable_lgns[pg_s];                  /* stable lgns for database synchronization */
```

```
        int stable_parts[pg_s];     /* stable lgn's partition sequence for database synchronization */
        int stable_count;                   /* number of stable lgns for database synchronization */
        int stability;                      /* flag to reach stability for database synchronization */
        int pair_number;            /* number of node pairs for this peer group */
        double active_time[part_s];                 /* LGN active time */
        double PTSERefresh_I;                       /* PTSE refresh interval */
        double PTSE_refresh[part_s][h_timer_s][3];          /* PTSE refresh  and expired timer */
        double PTSE_refresh1[part_s][h_timer_s][3];         /* PTSE refresh  and expired timer for
                                                                                    backup */
        struct LGN *next;                                   /* linked list */
} *first_lgn, *current_lgn, *lgn_ptr1, *lgn_ptr2, *lgn_ptr3, *lgn_ptr4, *lgn_ptr5, *lgn_ptr;
```

Similar to the data structure of the ATM switches, the network topology may have as many LGNs as needed. They are stored in a linked list pointed to by **current_lgn**. The first lgn is pointed to by **first_lgn**, next lgn is pointed to by **next**, and so on. That is, each lgn's pointer points to next lgn's pointer, if a lgn is added, the simulator program must create and initialize a lgn structure and put it on its linked list. Only the pointer for the first lgn is set to a fixed pointer, all other lgns are not set to a fixed pointer because the number of lgns for each network topology is not fixed. Note that, a lgn structure also represents a peer group information at the lowest level.

Some parameters are explained as follows:

address[add_len]	represents the ATM address of the first entry node for this peer group.
level	represents the node level of first entry node for this peer group.
part_s	represents the maximum number of partitions within one peer group.
partition	represents the partition number for this peer group. Normally, there is only one partition for one peer group.
temp_partition	represents the partition number for this peer group, because there may be several partitions when a link or node failure exists.
h_connect_s	represents the maximum number of higher level connections to other LGNs.
pg_s	represents the maximum number of lowest level peer groups.
h_data_s	represents the routing database information size for each LGN.
stable_lgns[pg_s]	used to record the stable lgns for database synchronization.
stable_parts[pg_s]	used to record the stable lgn's partition sequence for database synchronization because there may exist serveral partitions for this peer group.
h_timer_s	represents the maximum number of refresh timers (Nodal IG, IRA IG and Horizontal Links IG, etc.).

8.2 Physical Links

For the reader's convenience, the data structure used for the physical links is listed below:

```
struct Connection {
        char Hello_state[parallel_s][2];        /* Hello states for both nodes of parallel physical links */
        char DBS_state[2];                      /* DBS states for both nodes */
        int node[2];                            /* corresponding both nodes */
        int data[parallel_s][5][2];     /* transmitted data bidirectionally for parallel physical links */
        int DS_seq[2];                          /* DS sequence number for both nodes */
        int Master[2];                          /* Master flag for both nodes */
        int request[node_s][data_s][2];         /* PTSE request list for both nodes */
        int DBS_link;                           /* link seq for database synchronization */
        int link_number;                /* number of parallel physical links */
        int links[parallel_s][2];       /* link sequence for both nodes of parallel physical links */
        double delay[parallel_s];               /* parallel physical link delay */
        double up_time[parallel_s];             /* parallel physical link active/up time  */
        double up_time_min;             /* minimum value for parallel physical link active/up times */
        double Hello_timer[parallel_s][2][2];   /* Hello and Inactivity timer for both nodes of
                                                                parallel physical links */
        double DBS_timer;                       /* DBS timer for Master sending first DS packet */
        double Hello_timer1[parallel_s][2][2];  /* Hello and Inactivity timer for both nodes of
                                                        parallel physical links for backup */
        double DBS_timer1;                      /* DBS timer for Master sending first DS packet
                                                                for backup */
        struct Connection *next;                /* linked list */
} *first_connection, *current_connection, *connection_ptr;
```

Note that, there may be more than one physical link between any two nodes, therefore one connection is defined to represent parallel physical links between two nodes. Similar to the data structure of the ATM switches, the network topology may have as many connections as needed. They are stored in a linked list pointed to by **current_connection**. The first connection is pointed to by **first_connection**, next connection is pointed to by **next**, and so on. That is, each connection's pointer points to next connection's pointer, if a connection is added, the simulator program must create and initialize a connection structure and put it on its linked list. Only the pointer for the first connection is set to a fixed pointer. All other connections are not set to a fixed pointer because the number of connections for each network topology is not fixed.

In above data structure, some parameters are explained as follows:

parallel_s represents the maximum number of parallel physical links between two nodes.

data[parallel_s][5][2] used to store the transmitted data bidirectionally for different packet types (i.e., Hello, Database Summary, PTSE request, PTSP, PTSE Acknowledgment) for parallel physical links.

DBS_link represents the link seq for database synchronization, meaning that one parallel physical link between both nodes is used to perform the data synchronization mechanism, including the transmission of Database Summary, PTSE request, PTSP, PTSP Acknowledgment packets.

up_time_min represents the minimum value for parallel physical link active/up times, which is used as the connection's active time.

8.2.1 Higher level links

Additionally, if existing higher level links, the simulator program will utilize the following data structure for the higher level links:

```
struct H_connection {
        char Hello_state[outside_s][2];          /* SVCC-based RCC Hello states for both LGNs */
        char DBS_state[outside_s][2];            /* SVCC-based RCC DBS states for both LGNs */
        char Horizontal_state[outside_s][2];     /* Horizontal link states for both LGNs of multiple
                                                                              horizontal links */
        int DS_seq[outside_s][2];                /* DS sequence number for both LGNs */
        int Master[2];                           /* Master flag for both LGNs */
        int request[outside_s][pg_s][h_data_s][2];/* PTSE request list for both LGNs */
        int lgn[2];                              /* corresponding two LGNs */
        int RCC[outside_s][low_connect_s][2];    /* SVCC-based RCC */
        int low_connect_number[outside_s];   /* number of lowest level connections for the RCC */
        int link_number;                 /* number of multiple horizontal links */
        int links[outside_s][2];                 /* link sequence for both LGNs of multiple
                                                                          horizontal links */
        int add_flag[outside_s][2];      /* flag for AddInducingLink for multiple horizontal links */
        int part[outside_s][2];                  /* partition sequence for SVCC-based RCC and
                                                                          horizontal link */
        int SVCC_flag[outside_s];                /* link sequence for SVCC-based RCC located
                                                                          which horizontal link */
        int border_node[2*outside_s];          /* both border nodes for multiple horizontal links */
        int extension[2*outside_s][2];           /* LGN horizontal link extension IG */
        int stable_flag[outside_s][2];           /* flag for horizontal link stability */
        double active_time[outside_s];           /* SVCC-based RCC active time */
        double delay[outside_s];                 /* SVCC-based RCC delay */
        double processing_time[outside_s][5][2]; /* LGN processing overheads for different packet
                                                                                  types */
        double Hello_timer[outside_s][3][2];     /* SVCC-based RCC Hello timer */
        double DBS_timer[outside_s];             /* DBS timer for Master sending first DS packet */
        double Inactivity_timer[outside_s][2];   /* Horizontal link Inactivity timer */
        double Hello_timer1[outside_s][3][2];    /* SVCC-based RCC Hello timer for backup */
        double DBS_timer1[outside_s];            /* DBS timer for Master sending first DS packet
                                                                          for backup */
        double Inactivity_timer1[outside_s][2];  /* Horizontal link Inactivity timer for backup */
        struct H_connection *next;               /* linked list */
} *first_h_connection, *current_h_connection, *h_connection_ptr;
```

Note that, there may be two or more horizontal links between any two LGNs, therefore one higher level connection is defined to represent multiple horizontal links between two LGNs. Higher level connection is a general term, it contains the horizontal links and the SVCC-based RCC. In case of a partition one or more horizontal links may be specified as SVCC-based RCCs.

Similar to the data structure of physical links, the network topology may have several higher level connections when needed. They are stored in a linked list pointed to by **current_h_connection**. The first higher level connection is pointed to by **first_h_connection**, next higher level connection is pointed to by **next**, and so on. That is, each higher level connection's pointer points to next higher level connection's pointer, if a higher level connection is added, the simulator program must create and initialize a higher level connection structure and put it on its linked list. Only the pointer for the first higher level connection is set to a fixed pointer. All other higher level connections are not set to a fixed pointer because the number of higher level connections for each network topology is not fixed.

Some parameters are explained as follows:

outside_s represents the maximum number of node pairs as outside link between two peer groups.

Hello_state[outside_s][2]/DBS_state[outside_s][2] represents the SVCC-based RCC Hello/DBS states for both LGNs. Note that, normally there exists only one SVCC-based RCC between any two LGNs. However, due to partitioning resulting from a link or node failure, there may exist two or more partitions for one lowest level peer group.

DS_seq[outside_s][2] represents the DS sequence number for a pair of LGNs, in case of multiple SVCC-based RCCs due to partitioning.

request[outside_s][pg_s][h_data_s][2] represents the PTSE request list for both LGNs which considering possible multiple SVCC-based RCCs due to partitioning.

low_connect_s represents the maximum number of lowest level connections which compose the SVCC-based RCC.

RCC[outside_s][low_connect_s][2] represents the SVCC-based RCC which considering possible multiple SVCC-based RCC. Each SVCC-based RCC contains serveral lowest level connections and nodes, which are calculated by the SPF algorithm.

low_connect_number[outside_s] represents the number of lowest level connections for the RCC, which considers possible multiple SVCC-based RCCs.

stable_flag[outside_s][2] represents the horizontal link stability, that is, when the horizontal link Hello FSM reachs "2-WayReceived" state.

active_time[outside_s] represents the SVCC-based RCC active time which considers possible multiple SVCC-based RCC.

delay[outside_s] represents the SVCC-based RCC delay, which is the sum of related lowest level physical link delays.

processing_time[outside_s][5][2] represents the LGN processing overheads for different packet types which considers possible multiple SVCC-based RCC.

Before the SVCC-based RCC is established, the overhead is the sum of related lowest level node processing overheads. After the SVCC is established, the overhead only means the processing overhead of the node at the end of the SVCC-based RCC.

Hello_timer[outside_s][3][2] represents the SVCC-based RCC Hello timer which considers possible multiple SVCC-based RCC, including Hello Timer, Inactivity Timer, SVCIntegrity Timer and Horizontal Link Inactivity Timer for both LGNs.

9 Architecture of the Simulator

The simulator software has been designed in a modular fashion using a number of building blocks: initialization module, a user interface module, an event manager module, a number of protocol modules (i.e., Hello FSM, Neighboring Peer FSM, PGLE Election FSM, LGN related protocols), and control modules. These blocks interact with each other as shown on Figure 1.

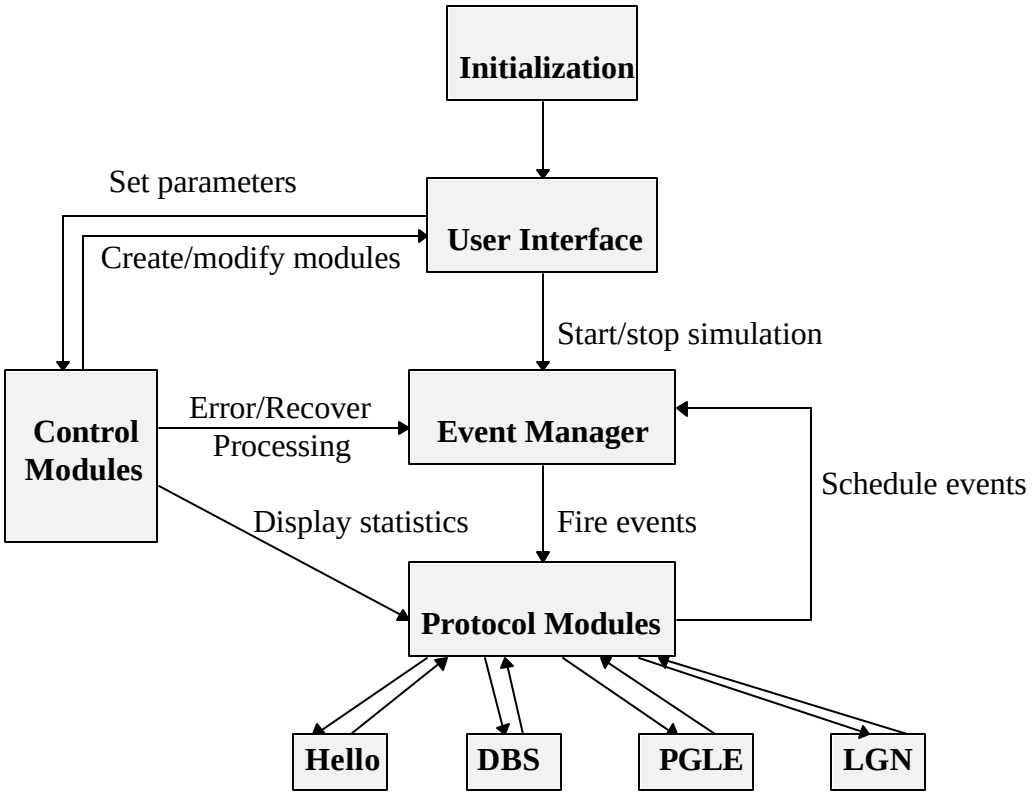


Figure 1: The building blocks of the simulator software

9.1 Initialization module

The initialization module, including the function **Initialize ()** and node and link related initialization part in the user interface function **User_interface ()**, is used to initialize the data structures and variables in the simulator software.

9.2 User Interface

The data structure used for the text user interface is as follows:

```
struct {
        char output;                    /* flag for output data per link each time or at the end */
        char failure;                   /* flag for link failure or node failure */
        char node_failure;              /* flag for physical node failure or logical node failure */
        int option[2];                  /* option for bundling PTSP packet and refreshing PTSE */
        int node_number;                /* total node number at the lowest level */
        int pg_number;                  /* peer group number at the lowest level */
        int connection_number;          /* connection number at the lowest level */
        int h_connection_number;        /* higher level connection number */
        int input;                      /* flag for input configuration file */
        int fail_pg;                    /* failed peer group for configuration */
        int fail_node[2];               /* failed nodes for configuration */
        int fail_connection;            /* failed connection for link failure for configuration */
        int fail_link_seq;                      /* failed link sequence for link failure for
configuration */
        int fail_h_connection;          /* failed higher level connection for configuration */
        double start_time;              /* simulation starting time */
        double execute_time;            /* simulation executed period */
        double failure_time;            /* when a link failure or node failure takes place */
        double restore_time;            /* when to restore the link failure or node failure */
} Inter;  /* user_interface  */
```

This text User Interface (UI) specifies the network topology and sets the parameters for simulation runs. The UI provides many options to cover many kinds of situations. It allows the user to setup different configurations for different nodes in order to accommodate the simulation for a variety of implementations. It also permits the user to define a link failure or node failure. Once a network topology is created and the appropriate parameters are set through the UI, the network can be simulated without the UI. The UI provides the control for the simulation start and stop.

The user interface (function **User_interface ()**) contains the following contents:

1) Setup options for PTSP bundling during database synchronization and for PTSE refreshing. Options for other packet types (Database Summary, PTSE request, etc.) are not considered.

2) Use the default node processing overheads and refresh interval or change the current default values, if necessary.
3) Set the total number of nodes for the PNNI network topology.
4) Setup the node (**ATM Switch**) configuration by using the ATM address, Node Level, Peer Group Leadership Priority, node active/up time, and node processing overheads for different packet types, if necessary.
5) Repeat step 4) until accomplished for all nodes' configuration in this network topology.
6) Determine the node's connectivity (**Physical Link**). If there is a connection between this node and any other node, it is necessary to determine how many physical links exist and what is the physical link delay and the physical link acive/up time.
7) Repeat step 6) until accomplished for all nodes' connectivity configuration in this network topology.
8) Setup a link failure or node failure and detail it, if any.
9) Setup the program simulation period and output flag.

Note that, higher level connections are determined by the lowest level node's ATM address, node level and connectivity entries.

9.3 Control module

The control module is used to monitor the running process of the simulator. It can create or modify the network topology. It also permits the user to process the link failure or node failure and link restore or node restore at the defined time. Any link failure or node failure may lead to lose of connectivity between other nodes and the PGL. Also new information received from another node may change this node's choice of preferred PGL. Moreover, it outputs the significant messages (display statistics) so that the user can analyze the running results for the values of performance measures of interest. There is no separate routine for the control module, its features are completed in the **User_interface ()** as follows:

1) Make sure the correctness of the entered parameters, and then continue. If not, modify selected entered parameters or just rerun the program to enter all parameters.
2) Determine whether there is a link failure or node failure in the future after the program runs. If yes, set up the failure time. When the simulation time reaches the failure time, the program automatically accesses the failure details through the function **Failure_processing ()**.
3) If a link failure or node failure exists and the user needs to restore this failure, when the simulation time reaches the restore time, the program accesses the restore details through the function **Restore_processing ()**.
4) Setup the program simulation time, which specifies the running period.
5) Determine whether the program is to display the statistics at each time of event execution so that the user can analyze the running results for the values of performance measures of interest. The transmitted data is counted at both directions for each link.

9.4 Event manager module

The simulator is *event* driven. The event queue is a queue of events arranged in time order. After an event is fired, the event in the queue is removed.

For the readers convenience, the data structures used for the event manager are listed as follows:

```
struct Event {
        char event;                     /* this event */
        int flag[6];                    /* flags for this event, including connection, link sequence,
                                                        node, protocol and partitions */
        int Dbase[node_s][data_s];      /* database information at the lowest level */
        int Dbase1[pg_s][h_data_s];     /* database information at the higher level */
```

```
        int Dbase_flag;                 /* type of packet */
        int extension[2*outside_s];     /* LGN horizontal link extension IG */
        int timer_flag;                 /* flag for real event or timer event */
        int node[node_s];           /* nodes in one peer group for flooding */
        double stamp;                   /* time stamp */
        struct Event *last;             /* last event */
        struct Event *next;             /* next event */
} *first_event, *current_event, *final_event, *event_ptr, *current1_event, *up_event1, *down_event1,
*down_event2;

struct Exe_event {
        char event;                     /* executed event */
        int flag[6];                    /* flags for this event, including connection, link sequence,
                                                                node, protocol and partitions */
        int Dbase[node_s][data_s];      /* database information at the lowest level */
        int Dbase1[pg_s][h_data_s];     /* database information at the higher level */
        int Dbase_flag;                 /* type of packet */
        int extension[2*outside_s];     /* LGN horizontal link extension IG */
        int node[node_s];           /* nodes in one peer group for flooding */
        struct Exe_event *next;         /* next executed event */
} *first_exe_event, *current_exe_event, *current1_exe_event;

struct {
        char Hello;                     /* Hello FSM event */
        char DBS[4];                    /* Neighboring Peer FSM events */
        char PGLE[3];                   /* PGLE FSM events */
        char SVCC_temp;     /* temporary event for SVCC-based RCC Hello when PGL elected */
        char SVCC_Hello;                /* SVCC-based RCC Hello FSM event */
        char SVCC_DBS[3];               /* SVCC-based RCC Neighboring Peer FSM events */
        char Horizontal;                /* Horizontal link Hello FSM event */
} Future_event;

struct {
        double Hello;                   /* Hello event's time stamp */
        double DBS[4];                  /* Neighboring Peer events' time stamp */
        double PGLE[3];             /* PGLE events' time stamp */
        double SVCC_temp;               /* temporary time stamp for temporary SVCC event when
                                                                PGL elected */
        double SVCC_Hello;              /* SVCC-based RCC Hello event's time stamp */
        double SVCC_DBS[3];           /* SVCC-based RCC Neighboring Peer events' time stamp */
        double Horizontal;              /* Horizontal link Hello event's time stamp */
} Future_stamp;
```

In the above data structures, **struct Event** is used for all events. During the simulation period, the event queue may have many events, and they are stored in a bi-directional linked list pointed to by **current_event**. The first event is pointed to by **first_event**, the final event is pointed to by **final_event**; next event points to by **next**, last event is pointed to by **last**, and so on. That is, each event's pointer is pointed to next event's pointer and last event's pointer, if an event is added, the simulator program must create and initialize an event structure and put it on its bi-directional linked list. Only the pointers to the first and final event are set to a fixed pointer. All other events are not set to a fixed pointer because the number of events during the program

running period for each network topology is not the same, also note the first event and final event are dynamically changed. In order to insert an event into the event queue arranged in time order, a function **Event_insert ()** is called. As for the timer expiration events, if the timer is disabled, its event is removed from the event queue by calling function **Timer_event ()**. If the timer is restarted, its event based upon the new timer expired value is inserted into the event queue and its event based upon the old timer expired value is removed from the event queue. The simulator software supports a limited number of timers.

struct Exe_event is used for the executed events. During the simulator program running period, the executed event queue may have several executed events at the earliest occurrence time (i.e., take place within the range of time accuracy), and they are stored in a linked list pointed to by **current_exe_event**. The first executed event is pointed to by **first_exe_event**, next executed event is pointed to by **next**, and so on. That is, each executed event's pointer points to next executed event's pointer, if an executed event is added, the simulator program must create and initialize an executed event structure and put it on its linked list. Only the pointer for the first executed event is set to a fixed pointer. All other executed events are not set to a fixed pointer because the number of executed events at the earliest occurrence time may not be the same.

struct Future_event and **struct Future_stamp** are used to temporarily store the future events and time stamps for different protocol modules, and then they will be stored in the event queue (**struct Event**).

We use a *discrete-event, continuous-time* model as the simulation model, meaning that the events of interest occur at discrete points in time. Under a discrete-event model, the time parameter is conceptually continuous and can thus be arbitrarily increased. The discrete changes in systems states or events occur at any real time. The simulation software contains an event manager (EM) that provides a general facility to schedule and send or fire an event. The function of the EM is to maintain and update an event queue arranged in the time order. The EM module also maintains a global clock to keep track of the simulation time. The EM is a critical module in the simulator software. Its primary functionality is the control of event queuing, which includes the following functions.

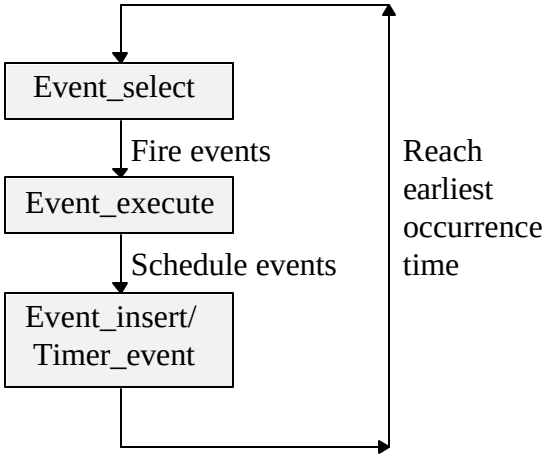


Figure 2: The building functions of event manager

1) **Event_select ().** This function is used to select the executed events with the earliest occurrence time in the event queue and return the selected number of executed events (**exe_number**). The selected events are put into the executed event buffer, and in the meanwhile, the selected events are removed from the event queue. Note that there may be more than one selected event with the earliest occurrence time.

2) **Event_execute (exe_number)**. This function is used to execute the selected events. The execution of event means to fire the event. When the selected event(s) are fired they are removed from the executed event queue. Furthermore, firing the event(s) may create one or more future events, which are then inserted into the event queue in the correct time order. The selected event related to different protocol modules will be executed by the different protocol modules, which are described below.

3) **Event_insert ()**. This function is used to insert the future events into the event queue in the correct time order. **Timer_event ()** is used to remove the old timer expiration event from the event queue.

4) The running clock hops to the earliest occurrence time, that is, the time stamp for the first event in the event queue. This approach is used to save CPU resources. The heart of the simulator is a loop executing the above cycle until the simulation is terminated.

Note that, during the program's running period, **struct PNNI** is used as a common data structure, which is listed as follows:

```
struct {
        int event_number;           /* event number in event queue */
```

```
        int fail_pg;                /* failed peer group during failure period */
        int pre_fail_node;          /* predefined failed node during failure period */
        int fin_fail_node;/* final failed node during failure period */
        int l_error[3];             /* flag for link failure, physical or logical node failure
                                                                    during failure period */
        int pre_fail_lgn;           /* predefined failed LGN during failure period */
        int fin_fail_lgn;           /* final failed LGN during failure period */
        int h_error[2];             /* flag for higher level link failure, or LGN failure
                                                                    during failure period */
        int local_PGL;       /* local PGL to check whether the uplink PTSE has been changed */
        int remote_PGL;   /* remote PGL to check whether the uplink PTSE has been changed */
        int uplink_seq;    /* sequence number to check whether the uplink PTSE has been changed */
        double time_min; /* earliest occurrence time for executed events */
        double timer[7];            /* temporary timer expired values */
} PNNI;   /* simulation parameters, especially for node or link failure */
```

9.5 Protocol modules

The selected events with the earliest occurrence time are executed separately based on different protocol modules. Each protocol module consists of an action routine. The routine is called for each event that happens to the protocol.

The protocol modules include the following:

- **Hello Finite State Machine (FSM)**, which is used to execute its related Hello events, and trigger the related events for the Neighboring Peer FSM and PGL Election FSM.
- **Neighboring Peer FSM (DBS protocol)**, which is used to execute its database synchronization related events, and trigger the related events for the PGL Election FSM. Note that, during database synchronization, the PTSEs may be flooded or refreshed.
- **PGL Election FSM**, which is used to execute its PGLE related events, trigger the flooding of the PTSE with preferred PGL, and may incur the establishment of an SVCC-based RCC. If two PGLs have been separately elected in two PGs and there exist border nodes between these two PGs, then the connectivity between these two PGLs is determined and how many lowest level links and how many lowest level nodes are calculated based upon the Shortest Path First (SPF) algorithm.
- **SVCC-Based RCC Hello FSM**, which is used to execute its related Hello FSM, and trigger its related events for the SVCC-based RCC Neighboring Peer FSM. This protocol is very close to the protocol between lowest-level neighbors, and uses the same packet type.
- **SVCC-based RCC Neighboring Peer FSM (SVCC DBS protocol)**, which is used to execute its database synchronization related events. This protocol is also very close to the protocol between lowest-level neighbors. Note that, during the

neighboring peer FSM, the PTSEs may need to be flooded or refreshed; for the SVCC-based RCC, we assume that there is no PGL election.

- **Horizontal Link Hello FSM**, which is used to communicate and agree upon the horizontal links which they will mutually advertise. The events and states of all horizontal links to an LGN neighbor are determined from the information in a single LGN Horizontal Link Extension information group included in the Hellos sent over the SVCC-based RCC.

All these protocol modules share several data structures. The protocol-related information is stored in the node structure, link structure and event structure. All these protocol modules are briefly described as follows.

9.5.1 Hello FSM

The Hello-related protocol executed module is the following:

```
void Execute_Hello (event, node1, node2, connection1, link_seq, direction1, delay1,
                                                                    processing_time1[]);
        char event;                     /* executed event */
        int node1;                      /* one node for the executed connection */
        int node2;                      /* another node for the executed connection */
        int connection1;                /* executed connection */
        int link_seq;                   /* physical link sequence between both nodes */
        int direction1;                 /* define the directionality of this connection */
        double delay1;                  /* delay for this physical link */
        double processing_time1[];      /* node processing overheads for different packet types */
```

This function will call the Hello FSM routine (**Hello_FSM()**) based upon the PNNI routing specification. Note that, the action taken in the **Hello_FSM ()** use the abstract style, for example:

```
switch (event) {
case 'A':           /* Link Up */
        switch (state) {
        case 'a': /* Down */
                if (link_flag == 0) Future_event.Hello = 'B';/* "1-Way Inside Received " */
                else Future_event.Hello = 'D';   /* link_flag = 1 or 2, "1-Way Outside Received" */
                Future_stamp.Hello = PNNI.time_min + processing_time1[0] + delay1;
                jitter = Random_jitter ();                  /* timer is jittered */
                PNNI.timer[0] = PNNI.time_min + Hello_I*(1.0 + jitter);
                state = 'b';                                /* "Attempt" state */
                break;
```

9.5.2 Neighboring Peer FSM

The data structure used for the neighboring peer FSM is the following:

```
struct {
        int packet[2];                      /* flag for different packet types */
        int lower1[node_s][data_s];         /* packet information for first event at the lowest level */
        int lower2[node_s][data_s];       /* packet information for second event at the lowest level */
        int db_number;                /* number of nodes in one node's database for PTSP separated */
        int higher1[pg_s][h_data_s];        /* packet information for first event at the higher level */
        int higher2[pg_s][h_data_s];      /* packet information for second event at the higher level */
        int flooding;                       /* flag for the PTSP flooding in database */
} DBS;  /* Neighboring Peer */
```

In order to make the executed neighboring peer FSM routine relatively independent for specific link and node, some temporary variables and arrays are used.

Similarly the DBS-related protocol executed module is the following:

```
void Execute_DBS (event, node1, node2, connection1, link_seq, direction1, delay1,
                                                                    processing_time1[]);
        char event;                         /* executed event */
        int node1;                          /* one node for the executed connection */
        int node2;                          /* another node for the executed connection */
        int connection1;                    /* executed connection */
        int link_seq;                       /* physical link sequence between both nodes */
        int direction1;                     /* define the directionality of this connection */
        double delay1;                      /* delay for this physical link */
        double processing_time1[];          /* node processing overheads for different packet types */
```

The execution of this function is quite complicated. It will call the DBS FSM routine **DBS_FSM ()**, because it involves the processing of a link failure or node failure and the flooding of some PTSP packets. If there exists a link failure or node failure after finishing the execution of related events (that is, call the routine, **Failure_processing ()**), the program may need to remove the failed link or the links attached to the failed node and to execute the function, **PGL_connectivity ()** to determine the connectivity to PGL. This is done for all nodes in this peer group, if a PGL exists for this peer group. The **PGL_connectivity ()** function utilizes the Shortest Path First (SPF) algorithm. If the failed link or node is to be restored, then when the simulation program reaches the restored time, it will call the routine, **Restore_processing ()**, to restore the failure and then continue running the program.

If there are more than two nodes in one peer group, during the database synchronization (including PGL election) and PTSE refreshing, the PTSP packets may need to be flooded everywhere in the peer group, therefore the DBS-related protocol executed module needs to call the flooding routine. For the programmer's convenience, the flooding routine is separated into two functions: **Flooding ()** and **Flooding_1 ()**. The **Flooding ()** is used to flood the PTSP packets during the database synchronization; the **Flooding_1 ()** is used to flood the PTSP packets for the PGL election and PTSE refreshing. In more detail, the **Flooding ()** and **Flooding_1 ()** are shown/defined as follows:

```
void Flooding (node1, connection1)
        int node1;                  /* one node for the executed connection */
        int connection1;            /* executed connection */

void Flooding_1 (node1, connection1, packet_flag)
        int node1;              /* one node for the executed connection */
        int connection1;        /* executed connection */
        int packet_flag;        /* packet type for PGLE and PTSE advertised/refreshed */
```

When a PTSE refresh timer expires or a PTSE expired timer expires, the simulator program calls the function **PTSE_Refresh ()** or **PTSE_Expired ()**. Note that, normally there is no PTSE expired events. The **PTSE_Expired ()** is called only when a link failure or node failure exists. During the execution of **PTSE_Expired ()**, the function, **Connectivity_two**, may be called to determine the connectivity between any two nodes in one peer group.

Additionally, it is necessary to calculate the data transmitted in each link, therefore, the function **Data_count ()**, is called.

9.5.3 PGLE FSM

Similarly the PGLE-related protocol executed module is the following:

```
void Execute_PGLE (event, node1, processing_time1[]);
        char event;                 /* executed event */
        int node1;                  /* one node for the executed connection */
        double processing_time1[];  /* node processing overheads for different packet types */
```

The execution of this routine calls the PGLE FSM, **PGL_Election ()**. If two PGLs have been already elected for two peer groups with border nodes' connecting and both uplink PTSE packets generated by the border node have arrived at respective PGLs, and then the program can call the routine **LGN_connecitvity ()** to determine their PGLs' connectivity, i.e. the LGN's connectivity, which invokes the SVCC-based RCC Hello protocol. Similar to the routine **PGL_connectivity ()**, the **LGN_connectivity ()** also uses the Shortest Path First algorithm to determine how many lowest level connections and nodes between this two LGNs. The following is a listing of the routine **LGN_connectivity ()**:

```
void LGN_connectivity (node1, node2, h_connection)
        int node1;              /* one PGL for the higher level connection */
        int node2;              /* another PGL for the higher level connection */
        int link_seq;           /* link sequence number for the SVCC-based RCC */
```

9.5.4 SVCC-based RCC Hello FSM

The listing of the routine **Execute_SVCC_Hello ()** is the following:

```
void Execute_SVCC_Hello (char event, int lgn1, int lgn2, int h_connection1, int link_seq,
                int direction1, int part1, int part2, double h_delay1, double h_processing_time1[]);
        char event;             /* executed event */
        int lgn1;               /* one LGN for the higher level connection */
        int lgn2;               /* another LGN for the higher level connection */
        int h_connection1;      /* higher level connection */
        int link_seq;           /* link sequence number for this Hello protocol */
        int direction1;         /* define the directionality of this connection */
        int part1;              /* partition sequence for one LGN */
        int part2;              /* partition sequence for another LGN */
        double h_delay1; /* delay for the higher level connection */
        double h_processing_time1[];    /* LGN processing overheads for different packet types */
```

The execution of this routine is quite similar to lowest level Hello protocol. It will need to call the SVCC-based RCC Hello FSM, **SVCC_Hello_FSM ()**.

9.5.5 SVCC-based RCC Neighboring Peer FSM

The listing of the routine **Execute_SVCC_DBS ()** is the following:

```
void Execute_SVCC_DBS (char event, int lgn1, int lgn2, int h_connection1, int link_seq,
        int direction1, int part1, int part2, double h_delay1, double h_processing_time1[]);
        char event;             /* executed event */
        int lgn1;               /* one LGN for the higher level connection */
        int lgn2;               /* another LGN for the higher level connection */
        int h_connection1;      /* higher level connection */
        int link_seq;           /* link sequence number for this DBS protocol */
        int direction1;         /* define the directionality of this higher level connection */
        int part1;              /* partition sequence for one LGN */
        int part2;              /* partition sequence for another LGN */
        double h_delay1; /* delay for the higher level connection */
        double h_processing_time1[];    /* LGN processing overheads for different packet types */
```

The execution of this routine is quite similar to lowest level neighboring peer FSM. It will need to call the SVCC-based RCC Neighboring Peer FSM, **SVCC_DBS_FSM ()**.

9.5.6 Horizontal link Hello protocol

The listing of the routine **Execute_Horizontal ()** is the following:

```
void Execute_Horizontal (char event, int lgn1, int lgn2, int h_connection1, int link_seq, int direction1,
                                int part1, double h_delay1, double h_processing_time1[]);
        char event;             /* executed event */
        int lgn1;               /* one LGN for the higher level connection */
        int lgn2;               /* another LGN for the higher level connection */
```

```
int h_connection1;          /* higher level connection */
int link_seq;               /* link sequence number for this DBS protocol */
int direction1;             /* define the directionality of this higher level connection */
int part1;                  /* partition sequence for one LGN */
double h_delay1; /* delay for the higher level connection */
double h_processing_time1[];        /* LGN processing overheads for different packet types */
```

The protocol for determining the state of horizontal links between LGNs is also based upon the Hello protocol. The execution of this routine will call the LGN Horizontal Link Hello Protocol, **Horizontal_FSM ()**.

Annex A. Modifications to Version 1.1 from Version 1.0.

This annex contains the detailed changes to the version 1.1 from the previous version 1.0. Since most of the changes were to the user interface, these changes are listed based on the user's perspective.

1) New command line

The previous command line contained only the executable or the executable and a seed value. However the new command line contains the executable and two or three values,
as previously stated in section 3.1, "Starting the Program"

aprops.exe *seed_value out_configuration_file* "or"
aprops.exe *seed_value output_configuration_file input_configuration_file*

The *seed_value* is an integer that is used as a input parameter to the random function, srand ( ).
The *output_configuration_file* is a file that will be created that will contain the user's network configuration as entered during the running of the user interface. This automates the creation of a network configuration file.
The *input_configuration_file* is a file that contains the user's network configuration. This permits the user to reuse a previous network configuration without the need to reenter the data.
Note that, the *output_configuration_file* and *input_configuration_file* can not use the same name.

2) Automatic creation of a network configuration file

The previous version required the user to input the network configuration manually through the user interface or manually offline. Version 1.1 automatically creates a network configuration file each time the program is invoked. This relieves the user from manual offline network configuration file creation.

3) Checking for valid input

The previous version 1.0 did not check for valid input. This created difficulty in fault isolation for network configuration data. The Version 1.1 does check for correct input of network configuration values. If entering data manually at the user interface, the input is tested and the user is prompted again, if the input is found to be in error. If an input_configuration_file is used and the input is tested and found to be in error the program terminates. For easy fault isolation the output_configuration_file contains the data that was accepted as correct. Using this information the user can easily find the value that is causing the error.

4) Correction of data

The previous version 1.0 allowed the user the option to reenter or correct the input data without terminating and then rerunning the program. The Version 1.1 requires that the user terminate and then rerun the program, if the data needs to be reentered or corrected.

5) Warning!!!

The program Version 1.1 will not work with previously created network configuration files using version 1.0.

Annex B. Modifications and Additions to Version 2.0 from Version 1.1.

This annex contains the detailed changes and additions to this version 2.0 from the previous version 1.1. These changes and additions are listed as follows:

1) Question of the number of nodes versus number of peer groups

The previous version 1.1 asked the user to enter the number of lowest level peer groups and the number of nodes in each peer group. Version 2.0 only requires the user to enter the total number of nodes in the PNNI network topology.

The previous version 1.1 asked the user to enter the connectivity by using the loop method, if the number of nodes in one peer group was less than 10, or by using the follow-up connected nodes otherwise. Version 2.0 asks the user about how many nodes or how many more nodes are connected for each node and then the user enters the node addresses for those nodes.

2) Input of ATM address

The previous version 1.1 used the simplified Peer group ID, Node ID and Node level format, i.e. general digital numbers. Version 2.0 utilizes the format specified in the PNNI specification, that is, the user needs to specify the node's ATM End System Address and node level. The Peer group ID will be determined by the node level and the prefix of ATM End System Address.

3) Input of higher level connections

The previous version 1.1 needed the user to differtiate the peer groups, that is, first to enter the parameters for each peer group in turn, and then to ask the connectivity between any two peer groups so as to create higher level connections. Version 2.0 only requires the user to enter the connectivity between any two nodes, the higher level connections will be determined by the program itself according the configuration information.

4) Execution of link or node failure

In the previous version 1.1, if a link failure or node failure existed, the program would halt at the failure time to allow the user to enter the specific link or node which was to fail. The program would also halt at the restore time to allow the user to enter the specific link or node which was to be restored. In Version 2.0, the link failure or node failure and its restoration is considered to be a part of configuration. Their details are configured as part of the network topology configuration file. The program will not halt at the failure or restore times.

5) Determination of Partitions

The previous version 1.1 did not check for partitioned peer groups. Version 2.0 gives the user the peer group partition information, if two or more partitions for one peer group exist. This

allows the user to determine whether the initial network topology configuration should consist of a partition peer group.

6) Processing of link failure or node failure

The previous version 1.1 only permitted a link failure to occur within one peer group. Version 2.0 allows a link failure to occur for any link.

7) Determination of Stability

The previous version 1.1 did not tell the user when the PNNI routing protocol reached stability, and what were the details. Version 2.0 outputs the stability information when the PNNI routing protocol reaches the completion of database synchronization and PGL election. The program will tell the user how quickly the failure information is flooded to all the nodes or the PNNI routing protocol recovers from a link failure or node failure.

8) Minor Changes

Version 2.0 modifies some names of variables in the data structures, and corrects some small errors or bugs in the program's design and implementation.

9) Warning!!!

Version 2.0 will not work with any previously created network configuration files using version 1.0 or 1.1.

Acknowledgements:
We would like to thank those who have registered and downloaded previous versions of the software. This provides us with critical data to continue the support of this project.

We would especially like to thank those who have used previous versions of this simulation software and provided us with their detailed problems, findings, studies, concerns, shortcomings and network configurations. These were very helpful in prioritizing, correcting and evolving this software.